**“ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ**

**УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ” (ТУСУР)**

Кафедра автоматизированных систем управления (АСУ)

Отчет по лабораторной работе №4

«Деревья»

По дисциплине СиАОДвЭВМ

Выполнил студент гр. 438-3:

\_\_\_\_\_\_\_\_\_\_\_\_\_ Канаев О.А.

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

Проверил

Доктор технических наук,

доцент каф. АСУ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Горитов А.Н

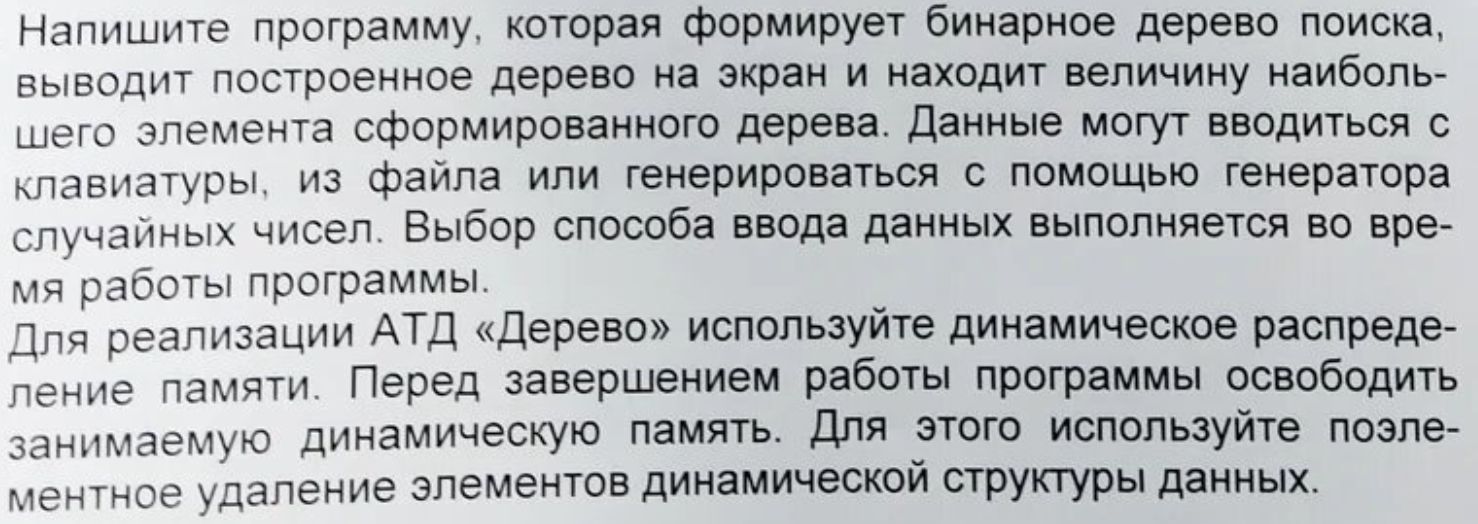
«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

Томск 2019

**Цель работы:**

Получить первичные навыки методов обработки абстрактных данных: деревья.

**Задание:**



**Алгоритм:**

1. Создать дерево (инициализировать)
2. Заполнение дерева: Из файла, рандомными числами и с клавиатуры
3. Поиск максимального эл-та дерева
4. Вывод дерева на экран
5. Удаление дерева

**Решение:**#include <iostream>

#include <locale.h>

#include <string>

#include <fstream>

#include <cstdlib>

#include <ctime>

using namespace std;

struct BTree

{

int data;

BTree \*left, \*right;

};

void DeleteTree(BTree \*&root);

void DelElem(BTree \*&root, int elem);

void Add(BTree \*&root, int num);

void Show(BTree \*&root, int level);

int Search(BTree \*&root, int SearchElement, int Koll);

int MaxElem(BTree \*&root);

int main()

{

system("chcp 1251");

BTree \*Root = new BTree;

if (Root == nullptr)

{

cout << "Память не выделилась";

return 0;

} //Проверка на выделение памяти

Root = nullptr;

int num;

cout << "Выберите:" << endl;

cout << "1-Из файла" << endl;

cout << "2-Случайное" << endl;

cout << "3-Ручной ввод" << endl;

cin >> num;

switch (num)

{

case(1):

{

ifstream file;

int number;

file.open("tree.txt");

if (!file.is\_open())

{

cout << "Ошибка открытия" << endl;

}

while (!file.eof())

{

file >> number;

Add(Root, number);

}

Show(Root, 0);

file.close();

break;

}

case(2):

{

srand(time(NULL));

int num, numElem;

cout << "Введите кол-во эл-в: " << endl;

cin >> numElem;

Show(Root, 0);

for (int i = 0; i < numElem; i++)

{

num = rand() % 100;

Add(Root, num);

}

Show(Root, 0);

break;

}

case(3):

{

int numElem, element;

cout << "Кол-во эл-в дерева: " << endl;

cin >> numElem;

for (int i = 0; i < numElem; i++)

{

cout << i + 1 << " элемент:";

cin >> element;

cout << endl;

Add(Root, element);

}

Show(Root, 0);

break;

}

}

cout << "Максимальный элемент дерева: " << MaxElem(Root) << endl;

DeleteTree(Root);

system("pause");

return 0;

}

void DeleteTree(BTree \*&root)

{

if (root != nullptr)

{

DeleteTree(root->left);

DeleteTree(root->right);

delete root;

root = nullptr;

}

}

void Add(BTree \*&root, int num)

{

if (root == nullptr) //Если корень пуст

{

root = new BTree;

if (root == nullptr) return; //Проверка на выделение памяти

root->data = num;

root->left = root->right = nullptr;

}

else

{

if (num < root->data) //Если элемент меньше, то идем в левый узел

{

if (root->left != nullptr)

Add(root->left, num);

else //Если левый узел пуст

{

root->left = new BTree;

if (root->left == nullptr) return; //Проверка на выделение памяти

root->left->left = root->left->right = nullptr;

root->left->data = num;

}

}

if (num > root->data) //Если элемент больше, то идем в правый узел

{

if (root->right != nullptr)

Add(root->right, num);

else //Если правый узел пуст

{

root->right = new BTree;

if (root->right == nullptr) return; //Проверка на выделение памяти

root->right->left = root->right->right = nullptr;

root->right->data = num;

}

}

}

}

void Show(BTree \*&root, int level)

{

if (root)

{

Show(root->right, level + 1);

for (int i = 0; i < level; i++)

cout << " ";

cout << root->data << endl;

Show(root->left, level + 1);

}

}

int Search(BTree \*&root, int SElem, int Koll)

{

if (root == nullptr)

{

return -1;

}

if (SElem < root->data)

{

Search(root->left, SElem, Koll + 1);

}

else

if (SElem > root->data)

{

Search(root->right, SElem, Koll + 1);

}

else

{

return Koll;

}

}

void DelElem(BTree \*&root, int elem) {

if (root == nullptr)

{

return;

}

if (elem < root->data)

{

DelElem(root->left, elem);

}

else

if (elem > root->data)

{

DelElem(root->right, elem);

}

else if ((root->left == nullptr) && (root->right == nullptr))

{

delete root;

root = nullptr;

}

else

{

DeleteTree(root);

}

}

int MaxElem(BTree \*&root) {

BTree \*ptr = root;

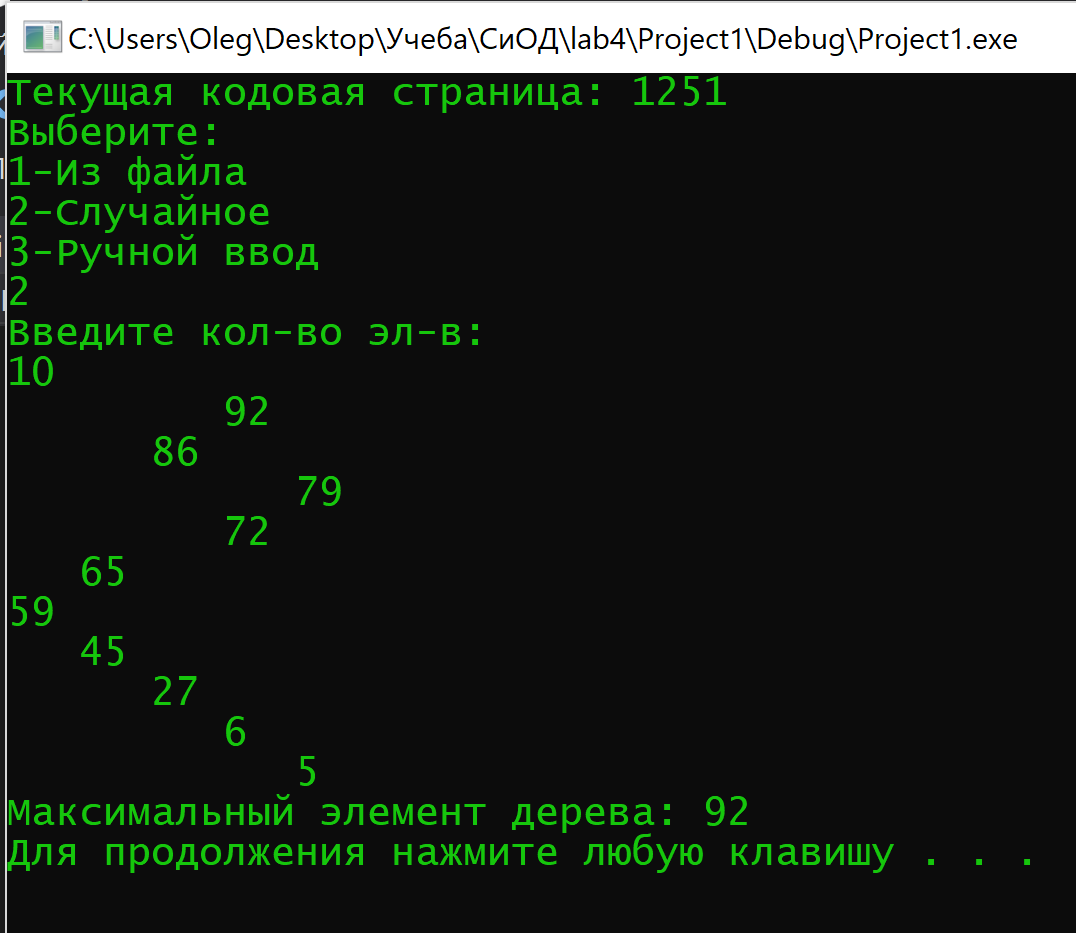
while (ptr->right != nullptr)

ptr = ptr->right;

return ptr->data;

}

**Результат работы:**



**Вывод:**Научился использовать абстрактный тип данных, дерево, с учетом плюсов и минусов. Деревья поиска, как и связные списки состоят из узлов, связанных указателями. Однако узлы эти упорядочены определенным образом — за счет этого увеличивается эффективность поиска, но осложняется операция вставки элемента.